

Automated Symbolic Analysis of ARBAC-Policies (Extended Version)

Alessandro Armando^{1,2} and Silvio Ranise²

¹ DIST, Università degli Studi di Genova, Italia

² Security and Trust Unit, FBK, Trento, Italia

Abstract. One of the most widespread framework for the management of access-control policies is Administrative Role Based Access Control (ARBAC). Several automated analysis techniques have been proposed to help maintaining desirable security properties of ARBAC policies. One limitation of many available techniques is that the sets of users and roles are bounded. In this paper, we propose a symbolic framework to overcome this difficulty. We design an automated security analysis technique, parametric in the number of users and roles, by adapting recent methods for model checking infinite state systems that use first-order logic and state-of-the-art theorem proving techniques. Preliminary experiments with a prototype implementations seem to confirm the scalability of our technique.

1 Introduction

Role Based Access Control (RBAC) [20] regulates access by assigning users to roles which, in turn, are granted permissions to perform certain operations. Administrative RBAC (ARBAC) [11] specifies how RBAC policies may be changed by administrators; thus providing support for decentralized policy administration, which is crucial in large distributed systems. For the sake of simplicity, we consider the URA97 component of ARBAC97 [19], which is concerned with the management of the user-role assignment by administrative roles. The generalization to other variants of ARBAC is left to future work.

As it is almost impossible for a human to foresee the subtle interplays between the operations carried out by different administrators because of the large number of possible interleavings. Automated analysis techniques are thus of paramount importance to maintain desirable security properties while ensuring flexible administration. Several techniques have been proposed, e.g., [17, 23, 22]. In general, security analysis problems are undecidable but become decidable under suitable restrictions. Indeed, the results of the analysis are valid under the assumptions that make them decidable. In this respect, one of the most severe limitations of the available techniques is that the number of users and roles is bounded, i.e. finite and known *a priori*. So, if one has proved that a certain property holds for, say, 1000 users and 150 roles and after some times, the number of

users or roles is changed for some reason, then the result of the previous analysis no more holds and the automated technique must be invoked again. It would be desirable to have analysis techniques capable of certifying that a certain property holds regardless of the number of users or roles so to make their results more useful.

In this paper, we propose a symbolic framework to specify ARBAC policies that enables the design of parametric (in the number of users and roles) security analysis techniques. The idea is to adapt recent techniques for model checking infinite state systems [14] that use decidable fragments of first-order logic and state-of-the-art theorem proving techniques to mechanize the analysis. The paper makes two contributions towards the goal of building parametric analysis techniques for ARBAC policies. The former is a **framework for the uniform specification of a variety of ARBAC policies**. In particular, we can describe security analysis problems where users and roles are finitely many but their exact number is not known *a priori*. The second contribution is a **symbolic** backward reachability **procedure** that can be used **to solve** an important class of security analysis problems, called **user-role reachability problems**, that allow one to check if certain users can acquire a given permission or, dually, if a user can never be given a role which would give him or her a permission which is not supposed to have. The security analysis problem is iteratively reduced to a series of satisfiability checks in a decidable fragment of first-order logic. We use ideas from model theory and the theory of well-quasi-ordering [14, 5] for the proof of termination of the method, which turns out to be the most substantial part of the proof of correctness. The decidability of the parametric goal reachability problem is obtained as a corollary of the correctness of the procedure.

Our decidability result is more general than those in [17, 23] which assume a bounded number of users and roles. A comparison with the result in [22] is more articulated. On the one hand, we are more general in allowing for a finite but unknown number of users and roles while in [22] the users are bounded and only the roles are parametric. On the other hand, we allow for only a restricted form of negation in the preconditions of certain administrative actions while [22] seems to allow for arbitrary negation. We plan to investigate how to extend our framework to allow for arbitrary negation in the near future while in this paper we focus on the core ideas. Finally, our procedure can consider several initial RBAC policies at the same time while [17, 23, 22] can handle only one.

Plan of the paper. In Section 2, we formally define ARBAC policies with their user-role reachability problem. In Section 3, we present our symbolic framework for the specification of ARBAC policies. In Section 4, we design a symbolic analysis procedures of ARBAC policies. In Section 5, we discuss some preliminary experiments with a prototype of our technique. Section 6 concludes and gives some hints about future work. The omitted proofs and some additional material can be found in the extended version of the paper [7].

2 RBAC and ARBAC policies

We assume familiarity with ARBAC (see, e.g., [11]) and many-sorted first-order logic with equality (see, e.g., [13]). Consider a signature Σ_{ARBAC} containing the sort symbols *User*, *Role*, and *Permission*, countably many constant symbols e_i^u, e_i^r, e_i^p (for $i \geq 0$) of sort *User*, *Role*, and *Permission*, respectively, the predicate symbols \succeq (written infix), pa , and ua of arity $Role \times Role$, $Role \times Permission$, and $User \times Role$, respectively, and *no function symbols*. A *RBAC policy* is a first-order structure $\mathcal{M} = (D, I)$ over this signature, where the interpretation of ua (in symbols, ua^I) is the user-role assignment relation, pa^I is the permission-role assignment, and \succeq^I is the role hierarchy. Without loss of generality, we consider structures that interpret the sort symbols into (disjoint) sets of users, roles, and permissions, respectively. Our notion of state corresponds to that of miniRBAC policy in [23].

An ARBAC policy prescribes how the user-role assignment, the permission assignment, and the role hierarchy of RBAC policies may evolve. As in [23] and according to the URA97 administrative control model [19], in this paper, we assume that the interpretations of \succeq and pa are constant over time and only that of ua may change. We also assume that \succeq^I is a partial order and refer to \succeq^I as the ‘more senior than’ relationship between roles. We abuse notation by denoting an interpretation $\mathcal{M} = (D, I)$ over Σ_{ARBAC} with the restriction s of I to ua when the rest of \mathcal{M} is clear from the context and write \succeq , pa , and ua instead of \succeq^I , pa^I , and ua^I (or $s(ua)$), respectively.

Let s be a RBAC policy. A user u is an *explicit member* of a role r in s if $(u, r) \in s(ua)$ or, equivalently, $s \models ua(u, r)$, where ‘ \models ’ is the standard satisfaction relation of many-sorted first-order logic. Similarly, u is an *implicit member* of r in s if $(u, r') \in s(ua)$ for some r' which is more senior than r or, equivalently, $s \models ua^*(u, r)$ where $ua^*(u, r)$ abbreviates the formula $\exists r'. (r' \succeq r \wedge ua(u, r'))$. Thus, u is *not* a member of r (neither implicit nor explicit) if for all role r' more senior than r , we have $(u, r') \notin s(ua)$ or, equivalently, $s \models \forall r'. (r' \succeq r \Rightarrow \neg ua(u, r'))$.

A *can_assign* action is a tuple $\langle r_a, C, r' \rangle$ such that r_a, r' are roles and C is a (possibly empty) finite set of *role expressions* of the form r or \bar{r} where r is a role. Sometimes, along the lines of [17], a set T of users can be attached to a *can_assign* action; in this case, users in T are assumed not to initiate any role assignment. A *can_revoke* action is a pair $\langle r_a, r' \rangle$ such that r_a, r' are roles. A user u *satisfies a role expression ρ in a RBAC policy s* if u is an implicit member of role r in s when ρ is r (or, equivalently, $s \models ua^*(u, r)$) and u is not a member of role r in s when ρ is \bar{r} (or, equivalently, $s \models \neg ua^*(u, r)$). A user u *satisfies the finite set $C = \{\rho_1, \dots, \rho_n\}$ of role expressions in a RBAC policy s* if u satisfies ρ_i in s , for each $i = 1, \dots, n$ ($n \geq 0$) or, equivalently, $s \models [\neg]ua^*(u, r_1) \wedge \dots \wedge [\neg]ua^*(u, r_n)$, where $[\neg]ua^*(u, r_i)$ denotes $ua^*(u, r_i)$ when ρ_i is r_i and $\neg ua^*(u, r_i)$ when ρ_i is \bar{r}_i . If $n = 0$, then $C = \emptyset$ and any user u always satisfies it. Let s, s' be two RBAC policies. A *can_assign* action $\langle r_a, C, r' \rangle$ is *enabled* in s if there exist users u_a, u such that u_a satisfies r_a in s and u satisfies C in s and s' is *obtained* from s by its application if $s'(ua) = s(ua) \cup \{(u, r')\}$. A *can_revoke* action $\langle r_a, r' \rangle$ is *enabled* in

s if there exists a user u_a such that u_a satisfies r_a in s and s' is obtained from s by its application if $s'(ua) = s(ua) \setminus \{(u, r')\}$. If α is a *can_assign* or a *can_revoke* action, we write $\alpha(s, s')$ to denote the fact that the action is enabled in s and s' is obtained from s by applying α . The pair (S_0, A) is an *ARBAC policy* when S_0 is a finite set of RBAC policies, called *initial*, and A is a finite set of *can_assign* and *can_revoke* actions. Let u be a user, RP be a finite set of pairs (r, p) where r is a role and p a permission. The pair $\gamma := (u, RP)$ is called the *goal* of the *user-role reachability problem* for $\Gamma := (S_0, A)$ which consists of answering the following question: is there a sequence s_0, \dots, s_m of states such that $s_0 \in S_0$, for each $i = 0, \dots, m-1$, there exists $\alpha \in A$ for which $\alpha(s_i, s_{i+1})$, $(u, r) \in s_m(ua)$, and $(r, p) \in pa$ for each pair $(r, p) \in RP$. If there is no such $m \geq 0$, then the goal γ is *unreachable*; otherwise, it is *reachable* and the sequence s_0, \dots, s_m of states is called a *run* leading Γ from an initial RBAC policy $s_0 \in S_0$ to a RBAC policy satisfying γ .

Example 1. We formalize the running example in [17]. Let \mathcal{M} be an RBAC policy such that $User := \{Alice, Bob, Carol\}$, $Permission := \{Edit, Access, View\}$, and $Role := \{Employee, Engineer, PartTime, FullTime, HumanResource, ProjectLead, andManager\}$.³ Every user is a member of role Employee. Managers work full-time. Project leaders are engineers. Alice is an engineer who is part-time. All employees have access permission to the office. Thus, \mathcal{M} is also such that $\succeq := \{(Engineer, Employee), (PartTime, Employee), (FullTime, Employee), (ProjectLead, Engineer), (Manager, FullTime)\}$, $pa := \{(Access, Employee), (View, HumanResource), (Edit, Engineer)\}$, $ua := \{(Alice, PartTime), (Alice, Engineer), (Bob, Manager), (Carol, HumanResource)\}$.

Examples of *can_assign* are: $\langle Manager, \{Engineer, FullTime\}, ProjectLead \rangle$, $\langle HumanResource, \emptyset, FullTime \rangle$, and $\langle HumanResource, \emptyset, PartTime \rangle$. The meaning of the first action is that a manager can assign a full-time engineer to be a project leader; the second and the third ones mean that a user in the human-resources department can turn any user to be full-time or part-time. If we attach to the previous assignments, the singleton set $T = \{Carol\}$ of users; then those actions cannot be performed by Carol even if she has the appropriate roles. Examples of *can_revoke* actions are: $\langle Manager, ProjectLead \rangle$, $\langle Manager, Engineer \rangle$, $\langle HumanResource, FullTime \rangle$, and $\langle HumanResource, PartTime \rangle$. For instance, the meaning of the first is that a manager can revoke the role of project leader to any user; the meaning of the other actions is similar. \square

3 Symbolic representation of ARBAC policies

Our framework represents (i) sets of RBAC policies as the models of a first-order theory whose signature contains only constant and predicate symbols but

³ For the sake of clarity, here and the other examples of the paper, we will abuse notation by using more evocative names for constants than e_i^α , $\alpha \in \{u, r, p\}$ ($i \geq 0$). Also, if constants have different identifiers, then they denote distinct elements. We use the same identifiers to denote constants and the elements they denote.

no function symbols, (ii) initial RBAC policies and constraints as universal formulae, and goals of reachability problems as existential formulae, and (iii) administrative actions (such as the *can_assign* and *can_revoke*) as certain classes of formulae. The assumptions on the three components allow us to design a decision procedure for the user-role reachability problem where the number of users and roles is finite but unknown. We now describe in details these assumptions.

Formal preliminaries. A Σ -theory is a set of sentences (i.e. formulae where no free variables occur) over the signature Σ . A theory T is axiomatized by a set Ax of sentences if every sentence φ in T is a logical consequence of Ax . We associate with T the class $Mod(T)$ of structures over Σ which are models of the sentences in T . A theory is *consistent* if $Mod(T) \neq \emptyset$. A Σ -formula φ is *satisfiable modulo T* iff there exists $\mathcal{M} \in Mod(T)$ such that \mathcal{M} satisfies φ (in symbols, $\mathcal{M} \models \varphi$). A Σ -formula φ is *valid modulo T* iff its negation is unsatisfiable modulo T and it is *equivalent modulo T* to a Σ -formula φ' iff the formula $(\varphi \Leftrightarrow \varphi')$ is valid modulo T . As notational conventions, the variables u, r, p and their subscripted versions are of sort *Users*, *Roles*, and *Permissions*, respectively; $\underline{u}, \underline{r}, \underline{p}$ denote tuples of variables of sort *Users*, *Roles*, *Permission*, respectively; $\varphi(\underline{x}, \underline{\pi})$ denotes a quantifier-free formula where at most the variables in the tuple \underline{x} may occur free and at most the predicate symbols in the tuple $\underline{\pi}$ may occur besides those of the signature over which φ is built. In this paper, we consider only consistent theories axiomatized by *universal sentences* of the form $\forall \underline{x}. \varphi(\underline{x})$. In the examples, we will make frequent use of the *theory of scalar values* v_1, \dots, v_n (for $n \geq 1$) of *type S* , denoted with $SV(\{v_1, \dots, v_n\}, S)$, whose signature consists of the sort S , the constant symbols v_1, \dots, v_n of sort S , and it is axiomatized by the following (universal) sentences: $v_i \neq v_j$ for $i, j = 1, \dots, n$, $i \neq j$, and $\forall x. (x = v_1 \vee \dots \vee x = v_n)$, where x is of sort S .

3.1 Symbolic representation of RBAC policies

Let T_{Role} be a Σ_{Role} -theory axiomatized by a finite set of universal sentences where Σ_{Role} contains the sort *Role*, the predicate \succeq , and countably many constants of sort *Role* but no function symbol. Let T_{User} be a Σ_{User} -theory axiomatized by a finite set of universal sentences where Σ_{User} contains the sort *User*, countably many constants of sort *User* but no function symbol. Let $T_{Permission}$ be a $\Sigma_{Permission}$ -theory axiomatized by a finite set of universal sentences where $\Sigma_{Permission}$ contains the sort *Role* and countably many constants of sort *Permission* but no function symbol. We emphasize that the signatures of these three theories may contain finitely many predicate symbols besides those mentioned above but no function symbols.

Example 2. For the version of ARBAC we are considering, the theory T_{Role} can be axiomatized by the following three universal sentences: $\forall r. (r \succeq r)$, $\forall r_1, r_2. ((r_1 \succeq r_2 \wedge r_2 \succeq r_1) \Rightarrow r_1 = r_2)$, and $\forall r_1, r_2, r_3. ((r_1 \succeq r_2 \wedge r_2 \succeq r_3) \Rightarrow r_1 \succeq r_3)$. This means that \succeq is interpreted as a partial order by the structures in $Mod(T_{Role})$. The set of basic roles and their positions in the partial order can be

defined, when considering Example 1, as the following sentences: $Engineer \succeq Employee$, $PartTime \succeq Employee$, $FullTime \succeq Employee$, $ProjectLead \succeq Engineer$, and $Manager \succeq FullTime$. The interested reader can see [7] for a discussion on how to formalize ARBAC with parametric roles.

For the theory T_{User} , we have a similar flexibility. For example, if there is only a *finite and known* number $n \geq 1$ of users, say e_1^u, \dots, e_n^u , then we can use the theory of a scalar value $SV(\{e_1^u, \dots, e_n^u\}, User)$. Another situation is when we have a *finite but unknown* number of users whose identifiers are, for example, linearly ordered (think of the integers with the usual order relation ‘less than or equal’). In this case, we add the ordering relation \leq of arity $User \times User$ to Σ_{User} and the following universal sentences constrain \leq to be a linear order: $\forall u.(u \leq u)$, $\forall u_1, u_2, u_3.((u_1 \leq u_2 \wedge u_2 \leq u_3) \Rightarrow u_1 \leq u_3)$, $\forall u_1, u_2.((u_1 \leq u_2 \wedge u_2 \leq u_1) \Rightarrow u_1 = u_2)$, and $\forall u_1, u_2.(u_1 \leq u_2 \vee u_2 \leq u_1)$. If $T_{User} = \emptyset$, then the identifiers e_i^u of users can be compared for (dis-)equality and there is again a *finite but unknown* number of users.

Similar observations also hold for $T_{Permission}$. Often, there is only a finite and known number of permissions that can be associated to roles. For example, continuing the formalization of Example 1, recall that we have only three permissions: Access, View, and Edit. So, $T_{Permission} := SV(\{Access, View, Edit\}, Permission)$. \square

As shown by the example above, the flexibility of our approach allows us to go beyond standard ARBAC policies by specifying the domains of users, roles, and permissions enjoying non-trivial algebraic properties which are useful to model, e.g., property-based policies [16]. We leave a detailed analysis of the scope of applicability of our framework to future work (as a first step in this direction, see [6]).

Now, we define $\Sigma_{ARBAC} := \Sigma_{Role} \cup \Sigma_{User} \cup \Sigma_{Permission} \cup \{pa, ua\}$ and let $T_{ARBAC} := T_{Role} \cup T_{User} \cup T_{Permission} \cup PA$, where PA is a set of (universal) sentences over $\Sigma_{Role} \cup \Sigma_{Permission} \cup \{pa\}$ characterizing the permission assignment relation.

Example 3. Consider again Example 1. The permission-role assignment is axiomatized by $PA := \{\forall p, r.(pa(p, r) \Leftrightarrow ((p = Access \wedge r = Employee) \vee (p = View \wedge r = HumanResource) \vee (p = Edit \wedge r = Engineer)))\}$. \square

Observe that a structure in $Mod(T_{ARBAC})$ over Σ_{ARBAC} is a RBAC policy.

3.2 Symbolic representation of initial RBAC policies, constraints, and goals

Since no axiom involving ua is in T_{ARBAC} , the interpretation of ua is arbitrary. We consider the problem of how to constrain the interpretation of ua by means of an example.

Example 4. We specify the user-role assignment of Example 1. Let T_{User}, T_{Role} , and $T_{Permission}$ be as in Example 3. Consider the formula $In(ua)$:

$$\forall u, r. (ua(u, r) \Leftrightarrow ((u = Alice \wedge r = PartTime) \vee (u = Alice \wedge r = Engineer) \vee (u = Bob \wedge r = Manager) \vee (u = Carol \wedge r = HumanResource))).$$

(Notice that $In(ua)$ can be seen as the Clark's completion [10] of the facts: $ua(Alice, PartTime)$, $ua(Alice, Engineer)$, $ua(Bob, Manager)$, and $ua(Carol, HumanResource)$.) It is easy to see that the interpretation considered in Example 1 satisfies $In(ua)$. \square

Since the formula $In(ua)$ used in the example above belongs to the class of universal sentences containing the state variable ua , we will use such a class of formulae, and denote it with \forall -formulae, to symbolically specify initial RBAC policies.

Example 5. Although in Example 4 the numbers of users and roles are fixed to certain values, our framework does not require this. For example, recall the discussion in Example 2 and take $T_{User} = \emptyset$, $T_{Role} = \emptyset$. Then, consider the following \forall -formula: $\forall u, r. (ua(u, r) \Leftrightarrow (u \neq e_0^u \wedge r \neq e_0^r))$. A RBAC policy s satisfying the formula is such that $(e_0^u, e_0^r) \notin s(ua)$ and $(e_i^u, e_j^r) \in s(ua)$ for every pair (i, j) of natural numbers with $i, j \neq 0$. Thus, there is no bound on the number of pairs (e_i^u, e_j^r) in $s(ua)$. \square

Notice that \forall -formulae are not only useful to describe initial RBAC policies but also to express constraints on the set of states that *can_assign* and *can_revoke* actions must satisfy. As an example, consider RBAC policies with separation of duty constraints, i.e. a user cannot be assigned two given roles. This can be enforced by using *static mutually exclusive roles* (SMER) *constraints* that require pairs of roles with disjoint membership (see, e.g., [23]). Formulae representing SMER constraints are \forall -formulae with the following form: $\forall u. \neg(ua(u, e_i^r) \wedge ua(u, e_j^r))$, for $i, j \geq 0$ and $i \neq j$. Notice that other kinds of constraints can be specified in our framework as long as they can be expressed as \forall -formulae.

Example 6. Let us consider again the situation described in Example 1. One may be interested in knowing if user Alice can take role FullTime and have permission Access. This property can be encoded by the following formula:

$$\exists u, r, p. (ua(u, r) \wedge pa(p, r) \wedge u = Alice \wedge r \succeq FullTime \wedge p = Access). \quad \square$$

Generalizing this example, we introduce \exists -formulae of the form $\exists \underline{u}, \underline{r}, \underline{p}. \varphi(\underline{u}, \underline{r}, \underline{p})$.

3.3 Symbolic representation of administrative actions

A *policy literal* is either $ua(u, r)$, $\neg ua(u, r)$, a literal over Σ_{User} (e.g., $u = e_i^u$ or $u \neq e_i^u$ for $i \geq 0$), or a literal over Σ_{Role} (e.g., $r = e_j^r$, $r \succeq e_j^r$, or their negations for $j \geq 0$). A *policy expression* is a finite conjunction of policy literals.

Administrative actions are represented by instances of formulae of the following form:

$$\exists u, r, u_1, r_1, r_2, \dots, r_k. (C(u, r, u_1, r_1, r_2, \dots, r_k) \wedge ua' = ua \oplus (u_1, e_i^r)) \quad (1)$$

$$\exists u, r, u_1. (C(u, r, u_1) \wedge ua' = ua \ominus (u_1, e_i^r)) \quad (2)$$

where $k, i \geq 0$, C is a policy expression called the *guard* of the transition, primed variables denote the value of the state variable ua after the execution of the transition, $ua \odot (u, e_i^r)$ abbreviates

$$\lambda w, v. (if (w = u \wedge v = e_i^r) then b else ua(w, v)),$$

and b is *true* when \odot is \oplus and it is *false* when \odot is \ominus .⁴ It is possible to symbolically represent *can_assign* actions as formulae of the form (1) and *can_revoke* actions as formulae of the form (2). We illustrate this with an example.

Example 7. We specify in our framework the administrative actions given in Example 1. The *can_assign* action $\langle Manager, \{Engineer, FullTime\}, ProjectLead \rangle$ corresponds to the following instance of (1):

$$\exists u, r, u_1, r_1, r_2. \left(\begin{array}{l} ua(u, r) \wedge r \succeq Manager \wedge u \neq Carol \wedge \\ ua(u_1, r_1) \wedge r_1 \succeq Engineer \wedge ua(u_1, r_2) \wedge r_2 \succeq FullTime \wedge \\ ua' = ua \oplus (u_1, ProjectLead) \end{array} \right).$$

Two observations are in order. First, the literal $u \neq Carol$ disables the transition when u is instantiated to *Carol*. This allows us to model the set $T = \{Carol\}$ of users that are prevented to execute assignments. Second, by simple logical manipulations and recalling the definition of the abbreviation ua^* introduced in Section 2, it is possible to rewrite the guard of the transition as $ua^*(u, Manager) \wedge ua^*(u_1, Engineer) \wedge ua^*(u_1, FullTime) \wedge u \neq Carol$. The simpler *can_assign* rules $\langle HumanResource, \emptyset, FullTime \rangle$ and $\langle HumanResource, \emptyset, PartTime \rangle$ can be specified by the following two instances of (1):

$$\begin{aligned} & \exists u, r, u_1. \left(\begin{array}{l} ua(u, r) \wedge r \succeq HumanResource \wedge u \neq Carol \wedge \\ ua' = ua \oplus (u_1, FullTime) \end{array} \right) \\ & \exists u, r, u_1. \left(\begin{array}{l} ua(u, r) \wedge r \succeq HumanResource \wedge u \neq Carol \wedge \\ ua' = ua \oplus (u_1, PartTime) \end{array} \right). \end{aligned}$$

Following [17], we call *AATU* (an abbreviation for ‘assignment and trusted users’) the set containing the above three formulae.

The *can_revoke* action $\langle Manager, ProjectLead \rangle$ is formalized by the following instance of (2): $\exists u, r. (ua(u, r) \wedge r \succeq Manager \wedge ua' = ua \ominus (u_1, ProjectLead))$. The remaining three *can_revokes* can be obtained from the formula above by simply replacing *Manager* and *ProjectLead* with *Manager* and *Engineer* for

⁴ We use λ -notation here for the sake of readability only. The same formulae can be easily recast in pure first-order logic. For example, (1) can be written as $\exists u, r, r_1, \dots, r_k. (C(u, r, r_1, \dots, r_k) \wedge \forall w, r. (ua'(w, r) \Leftrightarrow ((w = u \wedge r = e^r) \vee ua(w, r))))$.

$\langle \text{Manager}, \text{Engineer} \rangle$, with `HumanResource` and `FullTime` for $\langle \text{HumanResource}, \text{FullTime} \rangle$, and with `HumanResource` and `PartTime` for $\langle \text{HumanResource}, \text{PartTime} \rangle$. \square

Notice that the guards of the transitions of the form (1) do not correspond exactly to those introduced in Section 2. On the one hand, policy expressions give us the possibility to require a user u to be an explicit member of a certain role r in the guard of transition (by writing $ua^*(u, r)$) while preconditions of a *can_assign* can only require a user to be an implicit member of a role (i.e. $ua^*(u, r)$). On the other hand, it is not possible, in general, to express $\neg ua^*(u, r)$ (i.e. u is neither an explicit nor an implicit member of r), although it is possible to use $\neg ua(u, r)$ (i.e. u is not an explicit member of r). This is so because to express $\neg ua^*(u, r)$, a universal quantification is required; recall from Section 2 that $\neg ua^*(u, r)$ abbreviates $\forall r'. (r' \succeq r \Rightarrow \neg ua(u, r'))$. In other words, only a limited form of negation can be expressed in the guards of our formalization of a *can_assign* action. This simplifies the technical development that follows, in particular the proof of termination of the procedure used to solve the user-role reachability problem (see Section 4 for details). We plan to adapt a technique used in infinite state model checking for handling global conditions to allow $\neg ua^*(u, r)$ in the guards of transitions (see, e.g., [4]) but leave this to future work. Here, we observe that in many situations of practical relevance, it is possible to overcome this difficulty. For example, when there are only finitely many roles ranging over a set R , it is possible to eliminate the hierarchy as explained in [21] so that the framework proposed in this paper applies without problems. It is worth noticing that although the set of roles has been assumed to be bounded, our framework supports the situation where the set of users can be finite but its cardinality is unknown.

3.4 Reachability and satisfiability modulo T_{ARBAC}

At this point, it should be clear that the (algebraic) structures of users, roles, and permission can be specified by suitable theories; that we can symbolically represent RBAC policies and goals by using \forall -formulae and \exists -formulae, respectively, *can_assign* actions by formulae of the form (1), and *can_revoke* actions by formulae of the form (2). As a consequence, we can rephrase the user-goal reachability problem introduced in Section 2 as follows.

Let T_{ARBAC} be a Σ_{ARBAC} -theory given as described above and specifying the structure of users, roles, permission, role hierarchy, and the permission-role relation. If $\Gamma := (S_0, A)$ is an ARBAC policy together with a set \mathcal{C} of constraints on the set of states that the actions of the system must satisfy (e.g., SMER), then derive the associated *symbolic ARBAC policy* $\Gamma_s := (In(ua), Tr, C)$ as explained above, where In is a \forall -formula representing the initial set S_0 of RBAC policies, Tr is a finite set of instances of (1) or of (2) corresponding to the actions in A , and C is a finite set of \forall -formula representing constraints in \mathcal{C} . Furthermore, let

γ_s be an \exists -formula of the form

$$\exists u_1, r_1, p_1, \dots, u_n, r_n, p_n. \bigwedge_{i=1}^n (ua(u_i, r_i) \wedge r_i \bowtie e_{j_i}^r \wedge p_i = e_{j_i}^p), \quad (3)$$

called a *symbolic goal* and corresponding to a goal $RP := \{(e_{j_i}^r, e_{j_i}^p) \mid i = 1, \dots, n\}$, where $\bowtie \in \{=, \succeq\}$. Then, it is easy to see that the user-role reachability problem for Γ with RP as goal is solvable iff there exists a natural number $\ell \geq 0$ such that the formula

$$In(ua_0) \wedge \bigwedge_{i=0}^{\ell} (\iota(a_i) \wedge \tau(ua_i, ua_{i+1}) \wedge \iota(a_{i+1})) \wedge \gamma_s(ua_{\ell}) \quad (4)$$

is satisfiable modulo T_{ARBAC} , where τ is the disjunction of the formulae in Tr , and ι is the disjunction of those in C . Notice that the (big) conjunction over ℓ with In in (4) can be seen as a characterization of the set of states (forward) reachable from the initial set of states. Symmetrically (and more interestingly for the rest of this paper), the (big) conjunction over ℓ with γ_s in (4) characterizes the set of states backward reachable from the goal states. We observe that when $\ell = 0$, no actions must be performed and already some of the states in In satisfies γ_s , thus, formula (4) simplifies to $In(ua_0) \wedge \iota(ua_0) \wedge \gamma_s(ua_0)$.

Example 8. We illustrate the check for satisfiability of the formula (4) for $\ell = 0$ by reconsidering the situation described in Example 6. The problem was to establish if the formula $In(ua)$ of Example 4 and the goal formula of Example 6 are satisfiable modulo the theory T_{ARBAC} in Example 3. We assume that the set of constraints of the symbolic ARBAC policies is empty. In this context, the formula (4) above can be written as follows:

$$PO := \forall u, r. (ua(u, r) \Leftrightarrow \left(\begin{array}{ll} (u = Alice \wedge r = PartTime) & \vee \\ (u = Alice \wedge r = Engineer) & \vee \\ (u = Bob \wedge r = Manager) & \vee \\ (u = Carol \wedge r = HumanResource) & \vee \end{array} \right)) \wedge \exists u_1, r_1, p_1. (ua(u_1, r_1) \wedge pa(p_1, r_1) \wedge u_1 = Alice \wedge r_1 \succeq FullTime \wedge p_1 = Access) ,$$

where the existentially quantified variables in the goal have been renamed for clarity. The problem is to establish the satisfiability of PO modulo the theory T_{ARBAC} in Example 3. As it will be seen below, there exists an algorithm capable of answering this question automatically. For PO , the algorithm would return ‘unsatisfiable,’ entitling us to conclude that the set of initial states considered in Example 4 do not satisfy the goal of allowing Alice, who is a full-time employee, to get access to a certain resource. \square

If we were able to automatically check the satisfiability of formulae of the form (4), an idea to solve the user-role reachability problem for ARBAC policies would be to generate instances of (4) for increasing values of ℓ . However, this would not give us a decision procedure for solving the goal reachability problem but only

```

function BReach( $\Gamma : (In, Tr, C)$ ,  $\gamma : \exists$ -formula)
1   $P \leftarrow \gamma$ ;  $B \leftarrow false$ ;  $\tau \leftarrow \bigvee_{t \in Tr} t$ ;  $\iota \leftarrow \bigwedge_{i \in C} i$ ;
2  while ( $\iota \wedge P \wedge \neg B$  is satisfiable modulo  $T_{ARBAC}$ ) do
3      if ( $In \wedge P$  is satisfiable modulo  $T_{ARBAC}$ )
          then return reachable;
4       $B \leftarrow P \vee B$ ;
5       $P \leftarrow Pre(\tau, P)$ ;
6  end
7  return unreachable;

```

Fig. 1. The basic backward reachability procedure

a semi-decision procedure. In fact, the method terminates only when the goal is reachable from the initial state, i.e. when, for a certain value of ℓ , the instance of the formula (4) is unsatisfiable modulo T_{ARBAC} . When, instead, the goal is not reachable, the check will never detect the unsatisfiability and we will be forced to generate an infinite sequence of instances of (4) for increasing values of ℓ . In other words, the decidability of the satisfiability of (4) modulo T_{ARBAC} is only a necessary condition for ensuring the decidability of the user-role reachability problem. Fortunately, is possible to stop enumerating instances of (4) for a certain value $\bar{\ell}$ of ℓ when the formula characterizing the set of reachable states for $\ell = \bar{\ell} + 1$ implies that characterizing the set of reachable states for $\ell = \bar{\ell}$; i.e. we have detected a *fixed-point*. We explore this idea in the following section.

4 Symbolic analysis of ARBAC policies

A general approach to solve the user-role reachability problem is based on computing the set of backward reachable states. It is well-known that the computation of sets of backward (rather than forward) reachable states is easier to mechanize. For $n \geq 0$, the *n-pre-image* of a formula $K(ua)$ is a formula $Pre^n(\tau, K)$ recursively defined as follows: $Pre^0(\tau, K) := K$ and $Pre^{n+1}(\tau, K) := Pre(\tau, Pre^n(\tau, K))$, where⁵

$$Pre(\tau, K) := \exists ua'. (\tau(ua, ua') \wedge K(ua')). \quad (5)$$

The formula $Pre^n(\tau, \gamma)$ describes the set of states from which it is possible to reach the goal γ in $n \geq 0$ steps. At the n -th iteration of the loop, the *backward reachability algorithm* depicted in Figure 1, stores the formula $Pre^n(\tau, \gamma)$ in the variable P and the formula $BR^n(\tau, \gamma) := \bigvee_{i=0}^n Pre^i(\tau, \gamma)$ (representing the set of states from which the goal γ is reachable in *at most* n steps) in

⁵ In (5), we use a second order quantifier over the relation symbol ua , representing the state of the system. This should not worry the reader expert in first-order theorem proving since a higher-order feature is only used to give the definition of pre-image. We will see that we can compute a first-order formula logically equivalent to (5) so that only first-order techniques should be used to mechanize our approach.

the variable B . While computing $BR^n(\tau, \gamma)$, **BReach** also checks whether the goal is reachable in n steps (cf. line 3, which can be read as $In \wedge Pre^n(\tau, \gamma)$ is satisfiable modulo T_{ARBAC}) or a fixed-point has been reached (cf. line 2, which can be read as $\neg((\iota \wedge BR^n(\tau, \gamma)) \Rightarrow BR^{n-1}(\tau, \gamma))$ is unsatisfiable modulo T_{ARBAC} or, equivalently, that $((\iota \wedge BR^n(\tau, \gamma)) \Rightarrow BR^{n-1}(\tau, \gamma))$ is valid modulo T_{ARBAC}). Notice that $BR^{n-1}(\tau, \gamma) \Rightarrow BR^n(\tau, \gamma)$ is valid by construction; thus, if $((\iota \wedge BR^n(\tau, \gamma)) \Rightarrow BR^{n-1}(\tau, \gamma))$ is a logical consequence of T_{ARBAC} , then also $((\iota \wedge BR^n(\tau, \gamma)) \Leftrightarrow BR^{n-1}(\tau, \gamma))$ is so and a fixed-point has been reached. The invariant ι is conjoined to the set of backward reachable states when performing the fixed-point check as only those states that also satisfies the constraints are required to be considered. When **BReach** returns **unreachable** (cf. line 7), the variable B stores the formula describing the set of states which are backward reachable from γ which is also a fixed-point. Otherwise, when it returns **reachable** (cf. line 3) at the n -th iteration, there exists a run of length n that leads the ARBAC policy from a RBAC policy in In to one in γ . We observe that for **BReach** to be an effective (possibly non-terminating) procedure, it is mandatory that (i) the formulae used to describe the set of backward reachable states are closed under pre-image computation and (ii) both the satisfiability test for safety (line 3) and that for fixed-point (line 2) are effective.

Regarding (i), it is sufficient to prove the following result.

Property 1. Let K be an \exists -formula. If τ is of the form (1) or (2), then $Pre(\tau, K)$ is equivalent (modulo T_{ARBAC}) to an effectively computable \exists -formula.

Proof. Let $K(ua) := \exists \tilde{u}, \tilde{r}. \gamma(\tilde{u}, \tilde{r}, ua(\tilde{u}, \tilde{r}))$, where γ is a quantifier-free formula. By definition, $Pre(\tau, K)$ is $\exists ua'. (\tau(ua, ua') \wedge K(ua'))$ and there are two cases to consider. The former is when τ is of the form (1). In this case, $\exists ua'. (\tau(ua, ua') \wedge K(ua'))$ is equivalent to

$$\exists u, r, u_1, r_1, r_2, \dots, r_k. \left(C(u, r, u_1, r_1, r_2, \dots, r_k) \wedge \right. \\ \left. \exists \tilde{u}, \tilde{r}. \gamma(\tilde{u}, \tilde{r}, (ua \oplus (u_1, e^r))(\tilde{u}, \tilde{r})) \right)$$

by simple logical manipulations and recalling the definition of K . In turn, this can be expanded to

$$\exists u, r, u_1, r_1, r_2, \dots, r_k. (C(u, r, u_1, r_1, r_2, \dots, r_k) \wedge \\ \exists \tilde{u}, \tilde{r}. \gamma(\tilde{u}, \tilde{r}, (\lambda w, r. (if (w = u \wedge r = e^r) then true else ua(w, r)))(\tilde{u}, \tilde{r})))$$

by recalling the definition of \oplus . It is possible to eliminate the λ -expression by observing that each of its occurrence will be applied to a pair of existentially quantified variables from \tilde{u}, \tilde{r} so that β -reduction can be applied. After this phase, the ‘if-then-else’ expressions can be eliminated by using a simple case-analysis followed by the moving out of the existential quantifiers that allows us to obtain an \exists -formula. This concludes the proof of this case. The second case, i.e. when τ is of the form (2), is omitted because almost identical to the previous. \square

Observe also that $Pre(\bigvee_{i=1}^n \tau_i, K)$ is equivalent to $\bigvee_{i=1}^n Pre(\tau_i, K)$ for τ_i of forms (1) and (2), for $i = 1, \dots, n$.

Example 9. To illustrate Property 1, we consider one of the transitions written in Example 7 and the goal in Example 6. We compute the pre-image w.r.t. the second transition in *AATU* (where *HR* stands for *HumanResource* and *FT* for *FullTime*), i.e.

$$\begin{aligned} & \exists u, r, p. (ua'(u, r) \wedge pa(p, r) \wedge u = Alice \wedge r \succeq FT \wedge p = Access) \wedge \\ & \exists u_1, r_1, u_2. (ua(u_1, r_1) \wedge r_1 = HR \wedge u_1 \neq Carol \wedge ua' = ua \oplus (u_2, FT)) , \end{aligned}$$

where ua' is implicitly existentially quantified. By simple logical manipulations, we have

$$\begin{aligned} & \exists u, r, p, u_1, r_1, u_2. (pa(p, r) \wedge (\text{if } u = u_2 \wedge r = FT \text{ then true else } ua(u, r)) \wedge \\ & u = Alice \wedge r \succeq FT \wedge p = Access \wedge ua(u_1, r_1) \wedge r_1 = HR \wedge u_1 \neq Carol), \end{aligned}$$

which, by case analysis and some simplification steps, can be rewritten to

$$\begin{aligned} & \exists u, r, p, u_1, r_1, u_2. (pa(p, r) \wedge (r = FT \wedge u_2 = Alice \wedge p = Access \wedge \\ & ua(u_1, r_1) \wedge r_1 = HR \wedge u_1 \neq Carol) \vee \\ & (pa(p, r) \wedge u \neq u_2 \wedge ua(u, r) \wedge u = Alice \wedge r \succeq FT \wedge p = Access \wedge \\ & ua(u_1, r_1) \wedge r_1 = HR \wedge u_1 \neq Carol) \vee \\ & (pa(p, r) \wedge r \neq FT \wedge ua(u, r) \wedge u = Alice \wedge r \succeq FT \wedge p = Access \wedge \\ & ua(u_1, r_1) \wedge r_1 = HR \wedge u_1 \neq Carol)) , \end{aligned}$$

which is an \exists -formula according to Property 1. \square

Concerning the decidability of the satisfiability tests for safety and fixed-point in the backward reachability algorithm in Figure 1 (point (ii) above), we observe that the formulae at lines 2 and 3 can be effectively transformed to formulae in the form $\exists \underline{x} \forall \underline{y}. \varphi(\underline{x}, \underline{y}, ua)$ where \underline{x} and \underline{y} are disjoint, which belong to the *Bernays-Schönfinkel-Ramsey* (BSR) class (see, e.g., [18]). To see how this is possible, let us consider the formulae at line 2. This is the conjunction of a \forall -formula (ι), an \exists -formula (as discussed above, the variable P stores $Pre^n(\tau, \gamma)$, which by Property 1 is an \exists -formula), and another \forall -formula (as discussed above, the variable B stores $\bigvee_{i=0}^n Pre^i(\tau, \gamma)$ whose negation is a conjunction of \forall -formulae by Property 1, which is a \forall -formula). By moving out quantifiers (which is always possible as quantified variables can be suitably renamed), it is straightforward to obtain a BSR formula. Now, let us turn our attention to the formula at line 3. It is obtained by conjoining a \forall -formula (In is so by assumption) and an \exists -formula (stored in the variable P , see previous case). Again, by simple logical manipulations, it is not difficult to obtain a formula in the BSR class. We also observe that checking the satisfiability of BSR formulae modulo T_{ARBAC} can be reduced to checking the satisfiability of formulae in the BSR class since all the axioms of T_{ARBAC} are universal sentences, i.e. BSR formulae. Collecting all these observations, we can state the following result.

Property 2. The satisfiability tests at lines 2 and 3 of the backward reachability procedure in Figure 1 are decidable.

This property is a corollary of the decidability of the satisfiability of the BSR class (see, e.g., [18]). Example 9 above contains an illustration of a satisfiability test to which Property 2 applies.

4.1 Termination

The closure under pre-image computation (Property 1) and the decidability of the satisfiability checks (Property 2) guarantee the possibility to mechanize the backward reachability procedure in Figure 1 but do not eliminate the risk of non-termination. There are various sources of divergence. For example, the existential prefix of a pre-image is extended at each pre-image computation with new variables as shown in the proof of Property 1. Another potential problem is that the fixed-point could not be expressed by using disjunctions of \exists -formulae (according to line 4 in Figure 1) even if it exists so that the procedure is only able to compute approximations and thus never terminates. To show that both problems can be avoided and that the procedure in Figure 1 terminates, we follow the approach proposed in [14, 5] for proving the termination of backward reachability for certain classes of infinite state systems. We introduce a model-theoretic notion of certain sets of states, called *configurations*, which are the semantic counter-part of \exists -formulae, and then define a well-quasi-order on them: this, according to the results in [5], implies the termination of the backward reachability procedure. For lack of space, the full technical development is omitted and can be found in [7]; here, we only sketch the main ideas. We also point out that this result can be seen as a special case of that in [14], developed in a more general framework that allows for the formalization and the analysis of safety properties for concurrent, distributed, and timed systems as well as algorithms manipulating arrays. However, we believe worthwhile to prove termination for the procedure presented in this paper (along the lines of [14]) as some technical definitions become much simpler.

A *state of the symbolic ARBAC policy* $\Gamma := (In, Tr, C)$ is a structure $\mathcal{M} \in Mod(T_{ARBAC})$, i.e. it is an RBAC policy belonging to a certain class of first-order structures. A *configuration* of Γ is a state \mathcal{M} such that the cardinality of the domain of \mathcal{M} is finite. Intuitively, a configuration is a finite representation of a possibly infinite set of states that “contains at least the part mentioned in the configuration.” The following example can help to grasp the underlying intuition.

Example 10. As in Example 5, let $T_{User} = \emptyset$, $T_{Role} = \emptyset$. Consider the \exists -formula: $\exists u, r. (ua(u, r) \wedge u = e_0^u \wedge r = e_0^r)$. There is no bound on the number of pairs (e_i^u, e_k^r) in a RBAC policy s satisfying the \exists -formula above provided that $(e_0^u, e_0^r) \in s(ua)$. Our procedure for the reachability problem considers (only) those RBAC policies s of the form $s(ua) = \{(e_0^u, e_0^r)\} \cup \Delta$ where Δ is a (possibly empty) set of pairs (e_i^u, e_k^r) with $i, j \neq 0$. In other words, the procedure considers all those configurations which contain at least the pair (e_0^u, e_0^r) mentioned in the \exists -formula above plus any other (finite) set Δ of pairs. \square

The idea that a configuration represents a (possibly infinite) set of RBAC policies sharing a common (finite) set of user-role assignments can be made precise by using the notion of partial order. A *pre-order* (P, \leq) is the set P endowed with a reflexive and transitive relation. An *upward closed set* U of the pre-order (P, \leq) is such that $U \subseteq P$ and if $p \in U$ and $p \leq q$ then $q \in U$. A *cone* is an upward closed set of the form $\uparrow p = \{q \in P \mid p \leq q\}$. We define a *pre-order on configurations* as follows. Let \mathcal{M} and \mathcal{M}' be configurations of Γ ; $\mathcal{M} \leq \mathcal{M}'$ iff there exists an embedding from \mathcal{M} to \mathcal{M}' . Roughly, an embedding is a homomorphism that preserves and reflects relations (see [7] for a formal definition). A configuration is the semantic counter-part of an \exists -formula. Let $[[K]] := \{\mathcal{M} \in \text{Mod}(T_{\text{ARBAC}}) \mid \mathcal{M} \models K\}$, where K is an \exists -formula.

Lemma 1. *The following facts hold: (i) for every \exists -formula K , the set $[[K]]$ is upward closed and (ii) $[[K_1]] \subseteq [[K_2]]$ iff $(K_1 \Rightarrow K_2)$ is valid modulo T_{ARBAC} , for every pair of \exists -formulae K_1, K_2 .*

An upward closed set U is *finitely generated* iff it is a finite union of cones. A pre-order (P, \leq) is a *well-quasi-ordering (wqo)* iff every upward closed sets of P is finitely generated. This is equivalent to the standard definition of wqo, see [14] for a proof. The idea is to use only finitely generated upward closed sets as configurations so that their union is also finitely generated and we can conclude that the backward reachability procedure in Figure 1 is terminating because of the duality between configurations and \exists -formulae (Lemma 1).

Theorem 1. *The backward reachability procedure in Figure 1 terminates.*

As a corollary, we immediately obtain the following fact.

Theorem 2. *The user-role reachability problem is decidable.*

This result is more general than those in [17, 23] which assume a bounded number of users and roles. We are more general than [22] in allowing for a finite but unknown number of users and roles while in [22] the users are bounded and only the roles are parametric. However, we allow for only a restricted form of negation in the preconditions of *can_assign* actions while [22] seems to allow for arbitrary negation. Moreover, our procedure can consider several initial RBAC policies at the same time while [17, 23, 22] can handle only one.

Finally, notice that we can reduce other analysis problems (e.g., role containment) to user-role reachability problems and thus show their decidability. For lack of space, this can be found in [7].

5 Preliminary experiments

We briefly discuss some experiments with a prototype implementation of the symbolic reachability procedure in Figure 1 that we call **ASSA**, short for Automated Symbolic Security Analyser. We consider the synthetic benchmarks described in [23] and available on the web at [2] whereby both the number of users and roles is bounded. We perform a comparative analysis between ASSA

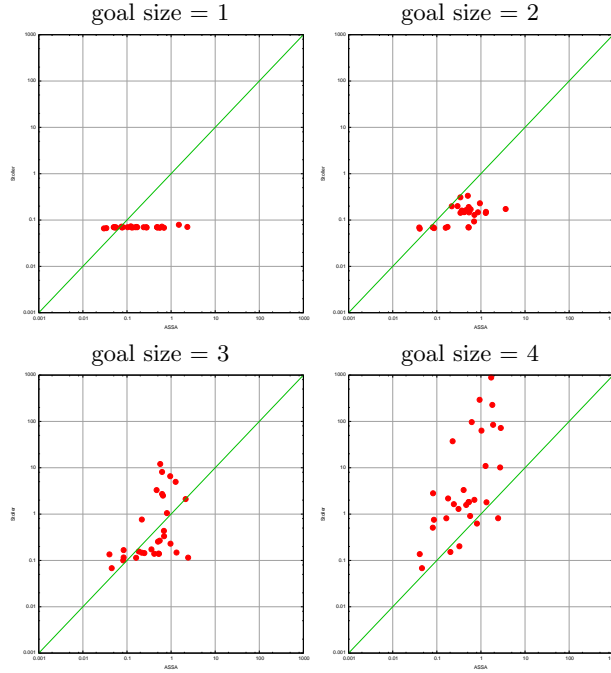


Fig. 2. Comparison between ASSA and **Stoller** on some benchmarks from [23, 2]

and the state-of-the-art tool in [23], called **Stoller** below. Our findings shows that **ASSA** scales better than **Stoller** on this set of benchmarks; the experiments were conducted on an Intel(R) Core(TM)2 Duo CPU T5870, 2 GHz, 3 GB RAM, running Linux Debian 2.6.32.

A client-server architecture is the most obvious choice to implement the proposed symbolic backward reachability procedure. The client generates the sequence of formulae representing pre-images of the formula representing the goal. In addition, the client is also assumed to generate the formulae characterising the tests for fix-point or for non-empty intersection with the initial set of policies. The server performs the checks for satisfiability modulo T_{ARBAC} and can be implemented by using state-of-the-art automated deduction systems such as automated theorem provers (in our case, SPASS [3]) or SMT solvers (in our case, Z3 [1]). Although these tools are quite powerful, preliminary experiments have shown that the formulae to be checked for satisfiability generated by the client quickly become very large and are not easily solved by available state-of-the-art tools. A closer look at the formulae reveals that they can be greatly simplified with substantial speed-ups in the performances of the reasoning systems. To this end, some heuristics have been implemented whose description is not possible here for lack of space; the interested reader is pointed to [6] for a complete description and more experiments.

We consider the randomly generated benchmarks in [2], where only the user-role assignment relation ua can be modified by *can_assign* or *can_revoke* actions (as assumed in Section 2). These benchmarks were generated under two additional simplifying assumptions: (i) a fixed number of users and roles, and (ii) absence of role hierarchy (this is without loss of generality under assumption (i) as observed in [23]). Besides the number of roles, one of the key parameter of the benchmarks (according to the parametrised complexity result derived in [23]) is the *goal size*, i.e. the number of roles in the set RP of a goal reachability problem (as defined at the end of Section 2) or, equivalently, the number of constants of sort *Role* occurring in the symbolic goal (3) of Section 3.4. The benchmarks are divided in five classes. The first and the second classes were used to evaluate the worst-case behavior of forward search algorithms (i.e. when the goal is unreachable) described in [23]. Our backward procedure (almost) immediately detects unreachability by realizing that no action is backward applicable. The fourth and fifth classes of benchmarks fix the goal size to one while the values of other parameters (e.g., the cardinality of the set R of roles) grow. In particular, the fourth class was used to show that the cost of analysis grows very slowly as a function of the number of roles while the fifth aimed to compare the performances of an enhanced version of the forward and the backward algorithms of [23]. For both classes, **ASSA** confirms that its running time grows very slowly according to the results reported in [23]. However, **ASSA** is slightly slower than **Stoller** because of the overhead of invoking automated reasoning systems for checking for fix-points instead of the *ad hoc* techniques of [23]. The most interesting class of problems is the third, which was used to evaluate the scalability of the backward reachability algorithm of [23] with respect to increasing values of the goal size 1, 2, 3, and 4. Figure 2 shows four scatter plots for values 1, 2, 3, and 4 of the goal size: the X and Y axes report the median times of **ASSA** and **Stoller**, respectively (logarithmic scale), to solve the 32 reachability problems in the third class of the benchmarks. A dot above the diagonal means a better performance of **ASSA** and viceversa; the time out was set to 1,800 sec. Although, both **Stoller** and **ASSA** were able to solve all the problems within the time-out, our tool is slower for goal sizes 1 and 2, behaves as **Stoller** for goal size 3, but outperforms this for goal size 4. These results are encouraging and seem to confirm the scalability of our techniques. For a detailed description of the implementation of **ASSA** and a more comprehensive experimental evaluation (confirming these results), the reader is pointed to [6].

6 Discussion

We have proposed a symbolic framework for the automated analysis of ARBAC policies that allowed us to prove the decidability of the parametric reachability problem. We used a decidable fragment of first-order logic to represent the states and the actions of ARBAC policies to design a symbolic procedure to explore the (possibly infinite) state space. Preliminary results with a prototype tool implementing the backward reachability procedure in Figure 1 are encouraging.

A detailed description of the implementation of the prototype and an extensive experimental analysis is available in [6].

There are two main directions for future work. First, it would be interesting to study to what extent other variants of ARBAC can be formalized in our framework, e.g., for UARBAC [16]. Second, we want to adapt techniques developed in the context of infinite state model checking to eliminate universal quantifiers in guards of administrative actions (called global conditions, see, e.g., [4]), to allow for unrestricted negation in *can_assigns*.

Acknowledgements. This work was partially supported by the “Automated Security Analysis of Identity and Access Management Systems (SIAM)” project funded by Provincia Autonoma di Trento in the context of the “team 2009 - Incoming” COFUND action of the European Commission (FP7), the FP7-ICT-2007-1 Project no. 216471, “AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures,” and the PRIN’07 Project 20079E5KM8 (Integrating automated reasoning in model checking: towards push-button formal verification of large-scale and infinite-state systems) funded by MIUR. Francesco Alberti must be thanked for his effort in implementing and benchmarking ASSA.

References

1. <http://research.microsoft.com/en-us/um/redmond/projects/z3>.
2. <http://www.cs.stonybrook.edu/~stoller/ccs2007>.
3. <http://www.spass-prover.org>.
4. P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite state processes with global conditions. In *Proc. of Computer Aided Verification (CAV)*, volume 4590 of *LNCS*, pages 14–157, 2007.
5. P. A. Abdulla and B. Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, pages 241–264, 2003.
6. F. Alberti, A. Armando, and S. Ranise. Efficient Symbolic Automated Analysis of Administrative Role Based Access Control Policies. In *Proc. of 6th ACM Symp. on Info., Computer and Comm. Security (ASIACCS’11)*, 2011.
7. A. Armando and S. Ranise. Automated Symbolic Analysis of ARBAC-Policies (Extended version). Available from <http://st.fbk.eu>, 2010.
8. M. Barletta, S. Ranise, and L. Viganò. Verifying the Interplay of Authorization Policies and Workflow in Service-Oriented Architectures. In *Proc. IEEE CSE’09, 12th Int. Conf. on Computational Science and Engineering, August 29-31, 2009*.
9. M. Y. Becker. Specification and Analysis of Dynamic Authorisation Policies. In *22nd IEEE Computer Security Foundations Symposium (CSF), IEEE, July 2009*.
10. K. Clark. *Logic and Databases*, chapter Negation as failure, pages 293–322. Plenum Press, New York, NY, 1978.
11. J. Crampton. Understanding and developing role-based administrative models. In *Proc. 12th ACM Conf. on Comp. and Comm. Security (CCS), pages 158–167, ACM Press, 2005*.
12. L. E. Dickson. Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors. *American J. of Math.*, 35(4):413–422, 1913.
13. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, Inc., 1972.

14. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT Model-Checking of Array-based Systems. In *Proc. of IJCAR*, LNCS, 2008.
15. W. Hodges. *Model Theory*. Cambridge University Press, 1993.
16. N. Li and Z. Mao. Administration in Role Based Access Control. In *Proc. ACM Symp. on Information, Computer, and Communication Security (ASIACCS)*, 2007.
17. N. Li and M. V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 9(4):391–420, 2006.
18. R. Piskac, L. de Moura, and N. Björner. Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. *J. of Autom. Reas.*, 44(4):401–424, 2010.
19. R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based control administration of roles. *ACM Transactions on Information and System Security (TISSEC)*, 1(2):105–135, 1999.
20. R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-Based Access Control Models. *IEEE Computer*, 2(29):38–47, 1996.
21. A. Sasturkar, P. Yang, S. D. Stoller, and C.R. Ramakrishnan. Policy analysis for administrative role based access control. In *Proc. of the 19th Computer Security Foundations (CSF) Workshop*. IEEE Computer Society Press, July 2006.
22. S. D. Stoller, P. Yang, M. I. Gofman, and C. R. Ramakrishnan. Symbolic Reachability Analysis for Parameterized Administrative Role Based Access Control. In *Proc. of. SACMAT'09*, pages 445–454, 2007.
23. S. D. Stoller, P. Yang, C.R. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control. In *Proc. of the 14th Conf. on Computer and Communications Security (CCS)*. ACM Press, 2007.

Plan of the Appendixes

We provide some additional material to illustrate and integrate the results presented in the paper:

- Appendix A discusses how to formalize parametric roles in our framework and explain that the decidability result for user-role reachability also cover this scenario.
- Appendix B presents the formal details of the termination of the backward reachability procedure in Figure 1.
- Appendix C discusses three related security analysis problems for ARBAC policies (namely, inductive policy invariant, role containment, and weakest preconditions) and their relationship with the user-role reachability problem.
- Finally, Appendix D describes in some detail an execution of the symbolic backward reachability procedure Figure 1 on a simple example taken from [23].

A Formalizing parametric roles

Here, we explain how it is possible to model ARBAC policies with parametrised roles as considered in, e.g., [22].

A role schema can be seen as an expression of the form $\rho(p_1, \dots, p_n)$ for $n \geq 0$, where ρ is a role name and p_i is a distinct parameter name $i = 1, \dots, n$. Each parameter can take values from a given data type containing an infinite number of values. An instance of a role schema is an expression of the form $\rho(p_1 = t_1, \dots, p_n = t_n)$, where t_i is a data value or a variable. For example, in the university policy considered in [22], the role schema $Student(dept, cid)$ is used for students registered for the course numbered cid offered by department $dept$, the role schema $Student(dept)$ is used for all students of a specific department $dept$, and the instance $Student(dept = cs, cid = 101)$ identifies students of the Computer Science department taking course 101. Role schemas can be overloaded by using parameter names; e.g., $Student$ can have one parameter named $dept$ or two parameters named $dept$ and cid . A parametrised version of ARBAC policies can use parametric roles to express role assignment and revocation in a very compact way. For example, in the case of the university policy, one can have the following role schemas: $Chair(dept)$, $Student(dept, cid)$, and $TA(dept, cid)$. Then, a *can_assign* rule is the following: the chair of department D (i.e. a user belonging to the role $Chair(dept = D)$) can assign a student of a department D taking course cs (i.e. a user belonging to the role $Student(dept = D, cid = CID)$) to be the teaching assistant of that course (i.e. a user belonging to the role $TA(dept = D, cid = CID)$).

In our symbolic framework, this situation can be formalized as follows. We introduce a predicate symbol extended with an extra argument for each parametric role, i.e. if the number of role names in the role schema ρ is n , then we use a predicate symbol ρ of arity $n + 1$ (this technique is standard for example to translate Entity-Relationship diagram schemas to fragments of first-order

logic). For the example above, we introduce the following predicate symbols: *Chair*, *Student*, and *TA* of arity 2, 3, and 3, respectively. We do not use parameter names, instead we fix an order on them so that we can use the standard way of building atoms in first-order logic. When a role schema is overloaded, we introduce a different predicate symbol in order to disambiguate the situation; a simple automated pre-processing phase can be used to eliminate overloading. In this context, the ‘can assign rule above can be written as follows:

$$\exists u, r, D, CID, u_1, r_1, r_2. \left(\begin{array}{l} Chair(D, r) \wedge ua(u, r) \\ Student(D, CID, r_1) \wedge ua(u_1, r_1) \\ TA(D, CID, r_2) \wedge ua' = ua \oplus (u_1, r_2) \end{array} \wedge \right),$$

where the variables r, r_1 , and r_2 are used as the names of the roles corresponding to the particular value of the attributes in the role schema. This means that we need to require that each relation is functional or, equivalently, that the interpretation of the predicate symbols must be partial functions. In our framework, this can be done by adding suitable formulae to the background theory T_{ARBAC} . For the example of the university policy considered above, we can simply write the following two \forall -formulae:

$$\begin{aligned} & \forall D, r_1, r_2. ((Chair(D, r_1) \wedge Chair(D, r_2)) \Rightarrow r_1 = r_2) \\ & \forall D, CID, r_1, r_2. ((Student(D, CID, r_1) \wedge Student(D, CID, r_2)) \Rightarrow r_1 = r_2) \\ & \forall D, CID, r_1, r_2. ((TA(D, CID, r_1) \wedge TA(D, CID, r_2)) \Rightarrow r_1 = r_2). \end{aligned}$$

Notice also that we can specify additional constraints among two or more relations if we can express them as \forall -formulae. It is not obvious how this feature can be added to the approach in [22]. For the example above, we have mentioned that we can have a role schema *Student(dept)* for identifying all students in the department *dept*. Indeed, *Student(dept, cid)* must characterize sub-sets of users of the role *Student(dept)*. If we introduce a binary predicate symbol *Student₁* of arity 2 corresponding to the role schema *Student(dept)*, then we can express this by the following \forall -formula:

$$\forall D, CID, r. (Student(D, CID, r) \Rightarrow Student_1(D, r)),$$

which can be added to T_{ARBAC} .

To summarize, our framework can handle parametrised roles as follows. First, the sub-theory T_{Role} of T_{ARBAC} becomes many-sorted: besides the sort *Role*, we introduce as many sort symbols—called *parameter sorts*—as domains for the parameters of each role. Furthermore, for each role symbol ρ of arity n , we introduce a predicate symbol of arity $n + 1$. Overloading is eliminated by introducing decorated versions of the predicate symbol and an order on the parameter names is fixed so that we can use the standard way of building atoms of first-order logic. Second, for each predicate symbol ρ of arity $n + 1$, we add the following functional constraint to T_{Role} and hence to T_{ARBAC} :

$$\forall \underline{x}, r_1, r_2. ((\rho(\underline{x}, r_1) \wedge \rho(\underline{x}, r_2)) \Rightarrow r_1 = r_2),$$

where \underline{x} is a tuple of length n of variables of appropriate sorts. If needed, we can add further constraints, (e.g., formalizing relationship between different role symbols) if these can be expressed as \forall -formulae. For example, it is worth noticing how to express the role hierarchy for parametrised role. Besides the usual axioms requiring \succeq to be a partial order, we can add also \forall -formulae of the following form:

$$\forall \underline{x}, \underline{y}, r_1, r_2. ((\rho_1(\underline{x}, r_1) \wedge \rho_2(\underline{y}, r_2)) \Rightarrow r_1 \succeq r_2),$$

where $\underline{x}, \underline{y}$ are tuples of variables of appropriate sorts, ρ_1, ρ_2 are two predicates representing parametric roles. This axiom requires that all instances of the parametric role ρ_1 are senior than those of role ρ_2 . Notice that one can design more sophisticated hierarchical relationships between role instances depending on the values of the parameters, provided that the signature is rich enough to express the constraints between the values of the parameters and that only \forall -formulae are used.

Finally, *can_assign* and *can_revoke* actions can be written by using existentially quantified variables ranging over the parameter names besides those ranging over users and roles; thus generalizing the shapes of actions (1) and (2). Formally, transitions have the following forms:

$$\begin{aligned} \exists u, r, u_1, r_1, r_2, \dots, r_k, \underline{p}. (C(u, r, u_1, r_1, r_2, \dots, r_k, \underline{p}) \wedge ua' = ua \oplus (u_1, e^r)) \\ \exists u, r, u_1, \underline{p}. (C(u, r, u_1, \underline{p}) \wedge ua' = ua \ominus (u_1, e^r)) \end{aligned}$$

where \underline{p} is a tuple of variables of parameter sorts and C is a constraint in which also literals built out of the predicate symbols introduced for modelling parametric roles may occur.

All the results proved in Sections 4 and 4 can be easily extended to cover ARBAC policies with parametric roles as soon as we observe that the formulae introduced here satisfy the assumptions on the theory T_{ARBAC} of Section 3.

B Termination of backward reachability

B.1 Pre- and well-quasi-orders: definitions and basic properties

A *pre-order* (P, \leq) is the set P endowed with a reflexive and transitive relation. We say that \leq is *decidable* if, given p_1 and p_2 in P , we can algorithmically check whether $p_1 \leq p_2$. An *upward closed set* U of the pre-order (P, \leq) is such that $U \subseteq P$ and if $p \in U$ and $p \leq q$ then $q \in U$. A *cone* is an upward closed set of the form $\uparrow p = \{q \in P \mid p \leq q\}$. An upward closed set U is *finitely generated* iff it is a finite union of cones.

For an upward closed set U , a *generator* of U is a set G such that (a) $U = \bigcup_{g \in G} \uparrow g$ and (b) $g_1 \leq g_2$ implies $g_1 = g_2$, for every $g_1, g_2 \in G$. It is easy to see that G contains only minimal elements (w.r.t. \leq) but, in general, it needs not to be unique. In any case, it is always possible to define a function $gen(U)$ returning a unique generator of U (the same chosen among the many possible ones).

A pre-order (P, \leq) is a *well-quasi-ordering* (*wqo*) iff every upward closed sets of P is finitely generated (this is equivalent to the standard definition of wqo, see [14] for a proof). In the case of a wqo, $\text{gen}(U)$ is finite because of property (b) of the definition of generator of U . This implies that every upward closed set U can be characterized by a finite set of configurations, namely $\text{gen}(U)$.

B.2 Some notions and results of model-theory

Let \mathcal{M} be a Σ -structure. A *substructure* of \mathcal{M} is a Σ -structure \mathcal{N} whose domain is contained in that of \mathcal{M} and such that the interpretations of the symbols of Σ in \mathcal{N} are restrictions of the interpretation of these symbols in \mathcal{M} ; conversely, we say that \mathcal{M} is a *superstructure* of \mathcal{N} . Let \mathcal{C} be a class of structures; we say that \mathcal{C} is *closed under substructures* if $\mathcal{M} \in \mathcal{C}$ and \mathcal{N} is a substructure of \mathcal{M} , then $\mathcal{N} \in \mathcal{C}$.

Property 3. A class \mathcal{C} of structures is closed under substructures iff there exists a theory T such that T contains only \forall -formulae and $\text{Mod}(T) = \mathcal{C}$.

A proof of this result can be found in any book on model theory, e.g., [15].

Let \mathcal{M} and \mathcal{N} two structures over the same signature Σ and M, N be their domains, respectively; an *embedding* s is an injective mapping from M to N such that (i) $s(f^{\mathcal{M}}(e_1, \dots, e_n)) = f^{\mathcal{N}}(s(e_1), \dots, s(e_n))$ for each function symbol f in the signature Σ and (ii) $(e_1, \dots, e_n) \in R^{\mathcal{M}}$ iff $(s(e_1), \dots, s(e_n)) \in R^{\mathcal{N}}$ for each predicate symbol R in Σ , where (e_1, \dots, e_n) is a tuple of elements in M of length equal to the arity of f or R , respectively. In other words, an embedding is a homomorphism that preserves and reflects relations. It is possible to show (see, e.g., [15]) that any embedding can be seen as the composition of an isomorphism followed by an “extension,” i.e. if there is an embedding from \mathcal{M} to \mathcal{N} , we can assume that \mathcal{M} is a substructure of \mathcal{N} (or dually, \mathcal{N} is a superstructure of \mathcal{M}).

Abstractly, (Robinson) diagrams give a logical formulation of model theoretic properties such as “there exists an embedding from structure \mathcal{M} to structure \mathcal{N} .” The importance of this will be clear when considering the definition of the pre-order on configurations (given in terms of the existence of an embedding between structures). Let \mathcal{M} be a Σ -structure and A be a sub-set of the domain of \mathcal{M} ; $\Sigma(A)$ is the signature obtained by adding to Σ new symbols of constants a for $a \in A$. We can regard \mathcal{M} as a $\Sigma(A)$ -structure when the interpretation function of \mathcal{M} is extended so that every element a in A is mapped to the constant a . The *(Robinson) diagram of A in \mathcal{M}* , in symbols $\delta_{\mathcal{M}}(A)$, is the set L of all $\Sigma(A)$ -literals such that $\mathcal{M} \models \ell$, for every $\ell \in L$.

Lemma 2 (Diagram Lemma). *Let \mathcal{M} and \mathcal{N} be two Σ -structures and M be the domain of \mathcal{M} . Then, there exists an embedding from \mathcal{M} to \mathcal{N} iff \mathcal{N} can be expanded to a $\Sigma(M)$ -structure which is a model of $\delta_{\mathcal{M}}(M)$.*

The proof of this fact is an immediate consequence of the definition of Robinson diagram given above and can be found in any book on model theory (see, e.g., [15]).

B.3 A pre-order on configurations: formal definition

Let Γ be a symbolic ARBAC policy, i.e.

$$\Gamma := (In(ua), \{\tau_1(ua, ua'), \dots, \tau_n(ua, ua')\}, \{\iota_1(ua), \dots, \iota_m(ua)\})$$

where In is a \forall -formula, ι_j is a \forall -formula, and τ_i is a transition formula of the forms (1) and (2).

Recall that a state of the ARBAC policy Γ is a structure $\mathcal{M} \in Mod(T_{ARBAC})$.

Definition 1. A configuration of Γ is a state \mathcal{M} where \mathcal{M} is a finite model, i.e. the cardinality of the domain of \mathcal{M} is bounded.

We are now in the position to define the pre-order on configurations.

Definition 2. Let \mathcal{M} and \mathcal{M}' be configurations. We write $\mathcal{M} \leq \mathcal{M}'$ iff there exists an embedding s from \mathcal{M} to \mathcal{M}' .

B.4 From \exists -formulae to configurations...

We show that \exists -formulae identify configurations. To state this result formally, we recall the following notation: $[[K]] := \{\mathcal{M} \in Mod(T_{ARBAC}) \mid \mathcal{M} \models K\}$, for K an \exists -formula.

Proposition 1. For every \exists -formula K , the set $[[K]]$ is upward closed.

Proof. Since the union of an upward closed set is still an upward closed set, we assume—without loss of generality—that $K(ua)$ is of the form $\exists \underline{r}, \underline{u}. \varphi(\underline{r}, \underline{u}, ua)$ where $\underline{u}, \underline{r}$ are tuples of variables for users and roles, respectively, and φ is a conjunction of literals (as we can always transform a Boolean combination of atoms into disjunctive normal form and then distribute the existential quantifiers over the disjunction). Under these assumptions, showing that $[[K]]$ is upward closed amounts to prove that if the configuration $\mathcal{M} \in [[K]]$ and $\mathcal{M} \leq \mathcal{N}$, then $\mathcal{N} \in [[K]]$, i.e. $\mathcal{N} \models K$. Now, assume that $\mathcal{M} \in [[K]]$ and $\mathcal{M} \leq \mathcal{N}$. This implies, by definition of $[[\cdot]]$, that (a) $\mathcal{M} \models K$ and (b) there exists an embedding s from \mathcal{M} to \mathcal{N} . From (a), by definition of truth, it follows that there exist tuples \underline{e}^u and \underline{e}^r of sort *User* and *Role*, respectively, such that $\mathcal{M} \models K(\underline{e}^u, \underline{e}^r)$. From (b) and the definition of embedding, we derive that

$$\mathcal{M} \models K(\underline{e}^u, \underline{e}^r) \text{ iff } \mathcal{N} \models K(s(\underline{e}^u, \underline{e}^r)).$$

The last two facts (and the well-known property that truth of quantifier-free formulae is preserved when considering superstructures) imply that $\mathcal{N} \models K$, as desired. This concludes the proof that $[[K]]$ is upward closed. \square

We show that entailment between \exists -formulae is equivalent to containment among configurations.

Proposition 2. $[[K_1]] \subseteq [[K_2]]$ iff $K_1 \Rightarrow K_2$ is valid modulo T_{ARBAC} , for every pair of \exists -formulae K_1, K_2 ,

Proof. There are two cases to consider. The ‘if’ case is trivial: it is an immediate consequence of the definition of truth. For the ‘only if’ case, we prove that if $K_1 \Rightarrow K_2$ is not valid modulo T_{ARBAC} , then $[[K_1]] \not\subseteq [[K_2]]$. Assuming that $K_1 \Rightarrow K_2$ is not valid modulo T_{ARBAC} is equivalent, by refutation, to say that $\neg(K_1 \Rightarrow K_2)$ (or $K_1 \wedge \neg K_2$) is satisfiable modulo T_{ARBAC} . In turn, this implies that $K_1 \wedge \neg K_2$ is satisfiable in a finite model according to the proof of the decidability of the BSR class. From this and Proposition 1, we can derive that $[[K_1]] \cap [[K_2]]^c \neq \emptyset$ (where c denotes the set complement operation). By simple set-theoretic manipulations, we derive $[[K_1]] \not\subseteq [[K_2]]$, as desired. \square

Lemma 1 is an immediate consequence of Propositions 1 and 2.

B.5 ... and viceversa: from configurations to \exists -formulae

We show that finitely generated upward closed sets of configurations are configurations of the form $[[K]]$, for some \exists -formula K . To do this, we use Robinson diagrams (introduced in Section B.2) since they give a logical formulation of model theoretic properties such as “there exists an embedding from structure \mathcal{M} to structure \mathcal{N} .” The importance of this is clear as soon as we recall the definition of pre-order over configurations that requires the existence of an embedding among structures to show that a configuration precedes another according to the pre-order. The main obstacle in using diagrams is that the formula $\delta_{\mathcal{M}}(M)$ usually contains infinitely many literals. Fortunately, in our case, it is possible to show that we can consider only a finite sub-set of literals in $\delta_{\mathcal{M}}(M)$ as all the others are implied by those in the sub-set.

Proposition 3. *The following facts hold:*

- (i) *with every configuration \mathcal{M} , it is possible to effectively associate an \exists -formula $K_{\mathcal{M}}$ (called diagram formula (for \mathcal{M})) such that $[[K_{\mathcal{M}}]] = \uparrow \mathcal{M}$,*
- (ii) *with every \exists -formula K , it is possible to effectively associate a finite set $\{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ of configurations such that K is equivalent to $\bigvee_{i=1}^n K_{\mathcal{M}_i}$,*
- (iii) *any finitely generated upward closed set of configurations coincides with $[[K]]$, for some \exists -formula K .*

Proof. We consider the three cases separately.

- (i) Let \mathcal{M} be a configuration and consider the “diagram” $\delta_{\mathcal{M}}(\underline{e}^u, \underline{e}^r)$ where \underline{e}^u and \underline{e}^r are finite tuples of users, roles, and permissions, respectively, that are also in the domain of \mathcal{M} .

Remark 1. Notice that $\delta_{\mathcal{M}}(\underline{e}^u, \underline{e}^r)$ is not the Robinson diagram as defined above; however, it turns out to be equivalent to $\delta_{\mathcal{M}}(\{e_i^u \mid i \geq 0\} \cup \{e_i^r \mid i \geq 0\})$, i.e. the “real” diagram. This is so because in any model of a BSR theory, there are only finitely many distinct atoms that “matter,” which are precisely those in $\delta_{\mathcal{M}}(\underline{e}^u, \underline{e}^r)$, because when checking for satisfiability we can always restrict to those constants that occur in the formula to be checked for satisfiability as discussed in the sketch of the proof of Property 2. (Recall,

in fact, that by applying Herbrand theorem, the Herbrand universe is finite and composed only of the constants occurring in the formula.) So, below, we refer to $\delta_{\mathcal{M}}(\underline{e}^u, \underline{e}^r)$ as the diagram and we treat it as the conjunction of its elements (i.e. as a first-order formula) since it is finite. \square

Now, take $K_{\mathcal{M}}$ to be the following \exists -formula: $\exists \underline{u}, \underline{r}. \delta_{\mathcal{M}}(\underline{u}, \underline{r})$. We are left with the problem of proving that $[[K_{\mathcal{M}}]] = \uparrow \mathcal{M}$. By the definitions of $[[K_{\mathcal{M}}]]$ and $\uparrow \mathcal{M}$, this is equivalent to show that a configuration \mathcal{N} is in $[[K_{\mathcal{M}}]]$, or—equivalently— $\mathcal{N} \models \exists \underline{u}, \underline{r}. \delta_{\mathcal{M}}(\underline{u}, \underline{r})$ iff $\mathcal{M} \leq \mathcal{N}$. Now, assume $\mathcal{N} \models \exists \underline{u}, \underline{r}. \delta_{\mathcal{M}}(\underline{u}, \underline{r})$, which is equivalent to $\mathcal{N} \models \delta_{\mathcal{M}}(\underline{e}^u, \underline{e}^r)$. By the Diagram Lemma (i.e. Lemma 2 above), this is equivalent to the existence of an embedding from \mathcal{M} to \mathcal{N} , which—in turn—is equivalent to $\mathcal{M} \leq \mathcal{N}$, by definition of \leq .

- (ii) Without loss of generality, we can assume K to be $\exists \underline{u}, \underline{r}. \bigvee_{k=1}^n \varphi_k(\underline{u}, \underline{r})$. For each $k = 1, \dots, n$, we can also assume (again without loss of generality) that there exists an existentially quantified variable x in $\underline{u} \cup \underline{r}$ such that $x = t$, for each constant in K . In this way, all the elements are explicitly mentioned in K . Now, in a BSR theory, every quantifier-free formula with at most m free variables is equivalent to a disjunction of the diagram $\delta_{\mathcal{M}}(X)$ where \mathcal{M} is a substructure of a model in the theory and X is a set of elements of cardinality at most m . Thus, K can be rewritten as

$$\bigvee_{\mathcal{A}} \exists \underline{u}, \underline{r}. \delta_{\mathcal{A}}(\underline{u}, \underline{r})$$

for \mathcal{A} ranging over the models whose cardinality is m (recall that the class of models of a BSR theory is closed under substructures). Each disjunct can be unsatisfiable, because it does not agree with the interpretation of ua , or satisfiable and, in this case, the model \mathcal{A} is a configuration such that $\exists \underline{u}, \underline{r}. \delta_{\mathcal{A}}(\underline{u}, \underline{r})$ is precisely $K_{\mathcal{A}}$, as desired.

- (iii) An immediate corollary of (i) and (ii) above. \square

The results in this and the previous subsection tells us that \exists -formulae and configurations can be used interchangeably.

B.6 Proof of termination of backward reachability

Theorem 1. The backward reachability procedure in Figure 1 terminates.

Proof. First of all, notice that when the algorithm return **reachable**, it also terminates (line 3). So, we consider the case when the goal is unreachable. Let $B(\tau, K) := \bigcup_{i \geq 0} [[BR^i(\tau, K)]]$. There two cases to consider.

- Let K be the \exists -formula given in input to the algorithm and assume that $B(\tau, K)$ is finitely generated (that $B(\tau, K)$ is an upward closed set is obvious because it is obtained as union of upward closed sets since $[[K]]$ is so by Proposition 1). Because of Proposition 2, we have that

$$[[BR^0(\tau, K)]] \subseteq [[BR^2(\tau, K)]] \subseteq \dots \subseteq [[BR^n(\tau, K)]] \subseteq [[BR^{n+1}(\tau, K)]] \subseteq \dots$$

Because $B(\tau, K)$ is finitely generated, we have that there exists n such that $[[BR^n(\tau, K)]] = [[BR^{n+1}(\tau, K)]]$ and, again by Proposition 2, we derive that $BR^n(\tau, K) \Leftrightarrow BR^{n+1}(\tau, K)$ is valid modulo T_{ARBAC} , i.e. the algorithm halts.

- Assume that the algorithm terminates. By Proposition 2, this is equivalent to $BR^n(\tau, K) \Leftrightarrow BR^{n+1}(\tau, K)$ is valid modulo T_{ARBAC} which, by Proposition 2, is equivalent to $[[BR^n(\tau, K)]] = [[BR^{n+1}(\tau, K)]]$, for some $n \geq 0$. Notice that $B(\tau, K) = [[BR^n(\tau, K)]]$ is finitely generated by Proposition 3.

So far, we have proved that the backward reachability procedure in Figure 1 terminates iff $B(\tau, K)$ is finitely generated. Thus, to conclude the proof, we show that $B(\tau, K)$ is indeed finitely generated. To this end, if we are able to prove that the pre-order on configurations is a wqo, then are entitled to conclude that $B(\tau, K)$ is finitely generated (recall the definition of wqo in Section B.1). Now, the pre-order on configurations is a wqo by Dickson’s Lemma [12]. In fact, a configuration is uniquely determined by a pair of integers counting the number of pairs (u, r) for which $ua(u, r)$ holds and the configuration ordering is obtained by component-wise comparison. This concludes the proof. \square

Combining the results above, we derive the main result of this paper, i.e. Theorem 2.

C Decidability of related security analysis problems

Here we consider three security analysis problems which are related to user-role reachability and discuss their decidability.

Inductive policy invariants. In [9, 8], the problem of checking properties that remain unaffected under any sequence of actions of arbitrary (but finite) length is considered. This is the dual problem of user-role reachability; in fact, it is not difficult to prove that if the backward reachability procedure terminates (with **unreachable**), then the fix-point is the strongest invariant. More precisely, In other words, a *policy invariant* is a formula which holds in every state of an ARBAC policy. In our framework, the problem of checking whether a property is an inductive invariant (a particular case of a policy invariant) turns out to be decidable because of Property 2. Let $\Gamma := (In, \{\tau_i\}, \{\iota_j\})$ be a symbolic ARBAC policy. The \forall -formula $\psi(ua)$ is an *inductive (policy) invariant for Γ* iff (a) $In(ua) \Rightarrow \psi(ua)$ is valid modulo T_{ARBAC} and (b) $(\iota(ua) \wedge \psi(ua) \wedge \tau(ua, ua')) \Rightarrow \psi(ua')$ is valid modulo T_{ARBAC} . It is easy to see that (a) and (b) can be reduced to the satisfiability of BSR formulae. In fact, (a) is equivalent to the unsatisfiability modulo T_{ARBAC} of $In(ua) \wedge \neg\psi(ua)$, which—in turn—can be transformed to a formula of BSR. Similarly, (b) is equivalent to the unsatisfiability modulo T_{ARBAC} of $\iota(ua) \wedge \psi(ua) \wedge \tau(ua, ua') \wedge \neg\psi(ua')$ which is again logically equivalent to a BSR formula. These observations with Property 2 imply the following fact.

Theorem 3. *The problem of establishing if a \forall -formula is an inductive policy invariant is decidable.*

Indeed, checking inductive invariants is a lot cheaper than running the backward reachability procedure. The drawback is that if a property ψ fails to be an inductive invariant, then we cannot conclude about its being an invariant of Γ (in other words, inductive invariants are a strict sub-class of policy invariants). However, we can take the complement $\neg\psi$ (which is an \exists -formula) of ψ and run the backward reachability procedure. If this returns **unreachable**, then we can conclude that ψ is an invariant of Γ .

Role containment. The problem of *role containment* for a symbolic ARBAC policy Γ consists of checking if every member of a certain role, say e_1^r , is also member of another role, say e_2^r , in every state reachable from the initial state. For simplicity, assume there is no role hierarchy. It is easy to reduce this to the user-role reachability problem by considering a role e_k^r not occurring in Γ and the following *can_assign* action:

$$\exists u, r, r_1. (ua(u, r) \wedge r = e_1^r \wedge \neg ua(u_1, r_1) \wedge r_1 = e_2^r \wedge ua' = ua \oplus (u, e_k^r)).$$

Let Γ' be obtained by adding the action above to Γ . It is easy to see that the role containment problem for Γ is solvable iff role e_k^r is reachable by Γ' .

Weakest precondition. The *weakest precondition problem* for a transition system Γ and goal γ consists of computing the minimal sets of initial role memberships of a given user e_k^u for which γ is reachable. This can be reduced to the user-role reachability problem by taking $\forall u, r. \neg ua(u, r)$ as the initial state formula In and then using a refinement of the backward reachability procedure in Figure 1. The refinement consists of using \exists -formulae whose matrix is a conjunction of literals only; this is without loss of generality as any \exists -formula can be transformed to a finite disjunction of \exists -formulae whose matrices are conjunctions of literals, called \exists^+ -formulae, by simple logical manipulations, and representing the search space by a forest of trees whose nodes are labelled by \exists^+ -formulae.

The root nodes are labelled by the \exists^+ -formulae whose disjunction is equivalent to the goal γ . Then, we iteratively extend each tree by selecting a node with no sons by adding as many sons as the number of \exists^+ -formulae which are equivalent to the pre-image of the formula labelling the father node. After the creation of a node n , we check whether a fix-point has been reached as follows. First, we consider the formula ψ labelling node n . Second, we take the disjunction of the \exists^+ -formulae labelling all the nodes in the tree except ψ : it is not difficult to see that this is equivalent to the content of the variable B of the procedure in Figure 1, i.e. it is the set of backward reachable states. Third, we check the satisfiability of $\neg((\iota \wedge \psi) \Rightarrow B)$, which is similar to the check at line 2 in Figure 1 except that ψ is an \exists^+ -formula instead of an \exists -formula. Because the pre-order on configurations is a wqo, it is possible to show that this procedure always terminates with finitely many trees. At this point, we collect all the \exists^+ -formulae labelling the nodes of the trees in the forest, compute the corresponding configurations (this is always possible because of Lemma 1, and take only those sets where the interpretation of ua has the minimal number of occurrences of

the user e_k^u as the first component. Since all the computation are effective, the procedure terminates.

By these reductions, we obtain the decidability of these two security analysis problems.

Theorem 4. *The containment and weakest precondition problems are decidable.*

D A worked-out example

We consider a simple reachability problem in [23]. There are several simplifying assumptions made by the authors of [23] that allow us to: (i) ignore permissions and focus only on roles, (ii) the role hierarchy can be abstracted away, (iii) there is just one administrative role and user capable of executing an administrative action of assignment, and (iv) there exists just one user to which administrative actions can be applied. As a consequence, a *can_assign* action can be seen as pair $\langle C, r' \rangle$ (where the administrative role has been omitted) while a ‘can revoke action only identifies the role r' to be revoked and ignore the administrative role that is supposed to apply the action, hence its specification will simply be $\langle r' \rangle$. Under these assumptions, the ARBAC policy considered in [23] consists of the following *can_assign* actions:

$$\begin{aligned} \text{can_assign}_1 &: \langle \{e_1^r\}, e_2^r \rangle, \\ \text{can_assign}_2 &: \langle \{e_2^r\}, e_3^r \rangle, \\ \text{can_assign}_3 &: \langle \{e_2^r\}, e_3^r \rangle, \\ \text{can_assign}_4 &: \langle \{e_3^r, \overline{e_4^r}\}, e_5^r \rangle, \\ \text{can_assign}_5 &: \langle \{e_5^r\}, e_6^r \rangle, \\ \text{can_assign}_6 &: \langle \{\overline{e_2^r}\}, e_7^r \rangle, \\ \text{can_assign}_7 &: \langle \{e_7^r\}, e_8^r \rangle, \end{aligned}$$

where we have dropped the numerical subscript of the constant e^u denoting a user because of assumption (iv); and the following ‘can revoke actions:

$$\begin{aligned} \text{can_revoke}_1 &: \langle r_1 \rangle, \\ \text{can_revoke}_2 &: \langle r_2 \rangle, \\ \text{can_revoke}_3 &: \langle r_3 \rangle, \\ \text{can_revoke}_4 &: \langle r_5 \rangle, \\ \text{can_revoke}_5 &: \langle r_6 \rangle, \\ \text{can_revoke}_6 &: \langle r_7 \rangle. \end{aligned}$$

The initial state s_0 of the ARBAC system is the following:

$$s_0(ua) := \{(e^u, e_1^r), (e^u, e_4^r), (e^u, e_7^r)\},$$

and the goal is to reach a state where the user e^u can be assigned to role e_6^r . As said in [23], the goal is not reachable from the initial state. Below, we explain how

to show that this is indeed the case in our framework and using the backward reachability procedure in Figure 1.

First of all, we specify the theory $T_{ARBAC} := T_{Role} \cup T_{User} \cup T_{Permission} \cup PA$ as follows:

$$\begin{aligned} T_{Role} &:= SV(\{e_1^r, \dots, e_8^r\}, Role) \\ T_{User} &:= SV(\{e^u\}, User) \\ T_{Permission} &:= \emptyset \\ PA &:= \emptyset. \end{aligned}$$

The formula $In(ua)$ characterizing the set of initial states is expressed by

$$\forall u, r. (ua(u, r) \Leftrightarrow \left(\begin{array}{l} (u = e^u \wedge r = e_1^r) \vee \\ (u = e^u \wedge r = e_4^r) \vee \\ (u = e^u \wedge r = e_7^r) \end{array} \right)).$$

The goal formula $\gamma(ua)$ characterizing the set of goal states is expressed by

$$\exists u, r. (ua(u, r) \wedge u = e^u \wedge r = e_6^r).$$

Notice that because of assumption (ii), the restricted form of negation allowed in the preconditions of transitions of the form (1) is sufficient to precisely describe the ‘can assign actions above:

$$\begin{aligned} can_assign_1 &: \exists u, r. (ua(u, r) \wedge r = e_1^r \wedge ua' = ua \oplus (u, e_2^r)) \\ can_assign_2 &: \exists u, r. (ua(u, r) \wedge r = e_2^r \wedge ua' = ua \oplus (u, e_3^r)) \\ can_assign_3 &: \exists u, r, r_1. \left(\begin{array}{l} ua(u, r) \wedge r = e_3^r \wedge \neg ua(u, r_1) \wedge r_1 = e_4^r \wedge \\ ua' = ua \oplus (u, e_5^r) \end{array} \right) \\ can_assign_4 &: \exists u, r. (ua(u, r) \wedge r = e_5^r \wedge ua' = ua \oplus (u, e_6^r)) \\ can_assign_5 &: \exists u, r. (\neg ua(u, r) \wedge r = e_2^r \wedge ua' = ua \oplus (u, e_7^r)) \\ can_assign_6 &: \exists u, r. (ua(u, r) \wedge r = e_7^r \wedge ua' = ua \oplus (u, e_8^r)). \end{aligned}$$

The *can_revoke* actions can be expressed as follows:

$$\begin{aligned} can_revoke_1 &: \exists u. (ua' = ua \ominus (u, e_1^r)) \\ can_revoke_2 &: \exists u. (ua' = ua \ominus (u, e_2^r)) \\ can_revoke_3 &: \exists u. (ua' = ua \ominus (u, e_3^r)) \\ can_revoke_4 &: \exists u. (ua' = ua \ominus (u, e_5^r)) \\ can_revoke_5 &: \exists u. (ua' = ua \ominus (u, e_6^r)) \\ can_revoke_6 &: \exists u. (ua' = ua \ominus (u, e_7^r)). \end{aligned}$$

Now, we can explain how the backward reachability procedure works on the example. In order to simplify the presentation, in the following, we use a variant of the backward reachability procedure in Figure 1. The differences are the following. First, instead of considering the disjunction of all the possible actions

and compute the pre-images of the goal with respect to this complex formula, we consider the pre-images of the goal with respect each possible action separately. Indeed, this allows us to write more compact formulae and, since it is easy to see that pre-image computation distributes over disjunction, it is sufficient to take the disjunction of the pre-images computed with respect to a single action to obtain the same formula computed by the procedure in Figure 1. Concerning the satisfiability checks, while the reachability test can be done as soon as we obtain a (satisfiable) pre-image with respect to a single action, the fix-point check requires a bit of care. In fact, after obtaining a (satisfiable) pre-image, the fix-point is *local* to that pre-image in the sense that all the (satisfiable) pre-images with respect to the remaining actions must also be checked for fix-point. Hence, a *global* fix-point is reached only when all the local fix-point are successful. Furthermore, each local fix-point check must be done by conjoining the actual pre-image with conjunction of the negation of each pre-image previously computed. It is not difficult to see that the global fix-point corresponds to the fix-point check of the procedure in Figure 1.

First of all, the backward procedures computes the pre-image of γ with respect to each *can_assign* and *can_revoke* actions. To illustrate how one of the pre-image computation is done, let us consider $Pre(can_assign_4, \gamma)$, i.e.

$$\begin{aligned} & \exists u_1, r_1. (ua'(u_1, r_1) \wedge u_1 = e^u \wedge r_1 = e_6^r) \wedge \\ & \exists u_2, r_2. (ua(u_2, r_2) \wedge u_2 = e^u \wedge r_2 = e_5^r \wedge ua' = ua \oplus (u_2, e_6^r)) \end{aligned}$$

where variables have been renamed to disambiguate the scope of applications of the existential quantifiers and ua' is implicitly existentially quantified. The formula can be rewritten as follows:

$$\exists u_1, r_1, u_2, r_2. \left(\begin{array}{l} ua'(u_1, r_1) \wedge u_1 = e^u \wedge r_1 = e_6^r \wedge \\ ua(u_2, r_2) \wedge u_2 = e^u \wedge r_2 = e_5^r \wedge \\ ua' = \lambda w, r. (if (w = u_2 \wedge r = e_6^r) then true else ua(w, r)) \end{array} \right)$$

by simple logical manipulations and recalling the definition of \oplus . Then, substituting the λ -expression we derive:

$$\exists u_1, r_1, u_2, r_2. \left(\begin{array}{l} \lambda w, r. (if (w = u_2 \wedge r = e_6^r) then true else ua(w, r))(u_1, r_1) \wedge \\ u_1 = e^u \wedge r_1 = e_6^r \wedge ua(u_2, r_2) \wedge u_2 = e^u \wedge r_2 = e_5^r \wedge \\ ua' = \lambda w, r. (if (w = u_2 \wedge r = e_6^r) then true else ua(w, r)) \end{array} \right)$$

which can be furtherly simplified as follows by using β -reduction:

$$\exists u_1, r_1, u_2, r_2. \left(\begin{array}{l} (if (u_1 = u_2 \wedge r_1 = e_6^r) then true else ua(u_1, r_1)) \wedge \\ u_1 = e^u \wedge r_1 = e_6^r \wedge ua(u_2, r_2) \wedge u_2 = e^u \wedge r_2 = e_5^r \wedge \\ ua' = \lambda w, r. (if (w = u_2 \wedge r = e_6^r) then true else ua(w, r)) \end{array} \right).$$

Now, we observe that $u_1 = u_2$ is valid modulo T_{ARBAC} since $T_{U_{ser}}$ constrains the set of users to be the singleton set $\{e^u\}$ and that $r_1 = e_6^r$ holds because it occurs in the formula above. Hence, we can simplify the formula above as follows:

$$\exists u_1, r_1, u_2, r_2. (u_1 = e^u \wedge r_1 = e_6^r) \wedge ua(u_2, r_2) \wedge u_2 = e^u \wedge r_2 = e_5^r$$

where ua' has been dropped since the equality $ua' = \lambda w, r.(\dots)$ is easily seen to be always satisfiable (this is so because to make the equality true, it is sufficient to take ua' equal to the λ -expression on the right). Finally, simple considerations on the quantified variables allow us to simplify the last formula even further so as to obtain:

$$\exists u, r. (ua(u, r) \wedge u = e^u \wedge r = e_5^r),$$

whose matrix is a policy constraint, exactly as the matrix of γ . This is not an accident as it is possible to show that the class of existentially quantified formulae whose matrix is a policy constraint are closed under pre-image computation. Let B_0 be γ and B_1 be the last formula above. The backward procedure performs a satisfiability check of the conjunction between In and B_1 , i.e. of the following formula:

$$\forall u, r. (ua(u, r) \Leftrightarrow \left(\begin{array}{l} (u = e^u \wedge r = e_1^r) \vee \\ (u = e^u \wedge r = e_4^r) \vee \\ (u = e^u \wedge r = e_7^r) \end{array} \right) \wedge \exists u, r. \left(\begin{array}{l} ua(u, r) \\ u = e^u \wedge r = e_5^r \end{array} \wedge \right)$$

so as to check whether the goal has been reached. Skolemizing the two existentially quantified variables, we obtain:

$$\forall u, r. (ua(u, r) \Leftrightarrow \left(\begin{array}{l} (u = e^u \wedge r = e_1^r) \vee \\ (u = e^u \wedge r = e_4^r) \vee \\ (u = e^u \wedge r = e_7^r) \end{array} \right) \wedge \left(\begin{array}{l} ua(\tilde{u}, \tilde{r}) \\ \tilde{u} = e^u \wedge \tilde{r} = e_5^r \end{array} \wedge \right),$$

where \tilde{r} and \tilde{u} are fresh constants. Now, observe that the universally quantified variable u can only take one value as we have assumed that the set of users contains just one element e^u ; hence it must be $\tilde{u} = e^u$. So, we are left with the problem of instantiating the universally quantified variable r . The decidability result of Property 2 allows us to consider only the instances of the formula where u is instantiated to e^u and r to \tilde{r} . It is not difficult to see that the resulting formula is unsatisfiable, thus entitling us to conclude that the sets of states characterized by B_1 and In are disjoint and the goal state is not reachable by applying can_assign_4 .

Then, the backward procedure proceeds to check for a fix-point. This is equivalent to the validity of $B_1 \Rightarrow B_0$ or to the unsatisfiability of its negation, namely $B_1 \wedge \neg B_0$:

$$\exists u, r. \left(\begin{array}{l} ua(u, r) \\ u = e^u \wedge r = e_5^r \end{array} \wedge \right) \wedge \forall u, r. \neg (ua(u, r) \wedge u = e^u \wedge r = e_6^r).$$

As before, we Skolemize the existentially quantified variables so as to obtain the following formula:

$$\left(\begin{array}{l} ua(\tilde{u}, \tilde{r}) \\ \tilde{u} = e^u \wedge \tilde{r} = e_5^r \end{array} \wedge \right) \wedge \forall u, r. \neg (ua(u, r) \wedge u = e^u \wedge r = e_6^r).$$

where \tilde{u}, \tilde{r} are fresh constants. As before, because of Property 2, without loss of generality, we can restrict to consider the formula obtained by instantiating

u to e^u and r to \tilde{r} : this time, however, we conclude that the formula is satisfiable. Thus, we have shown that a fix-point has not been reached and we need to compute the pre-images of B_1 w.r.t. the all the *can_assign* and *can_revoke* actions. However, before computing the pre-images of B_1 , we also need to compute the pre-images of B_0 w.r.t. τ in $\{can_assign_i | i = 1, 2, 3, 5, 6\} \cup \{can_revoke_i | i = 1, \dots, 6\}$, i.e. for the remaining assignments and revocations. This turns out to be useless as all the formulae obtained in this way characterizes sets of states that are sub-sets of those specified by γ or, in other words, we have reached a (local) fix-point. For the sake of conciseness, we do not do this here. However, the reader can verify this as a simple exercise by following the steps taken above for computing $Pre(can_assign_4, U)$ and checking for safety and fix-point. Similar observations hold also for the pre-images of B_1 : it turns out that all these formulae implies B_1 , i.e. several (local) fix-point have been reached, and are unsatisfiable when considered in conjunction with In , i.e. they pass the safety check. As a consequence, we can conclude that we have reached a (global) fix-point and the goal is not reachable.